LINK TO NOTEBOOK: https://colab.research.google.com/drive/1LYPieuBGzzBd7yb0-CNvxFe3EgvLMeB4?usp=sharing

In this homework fit and select a classifier to predict credit card default using `default_of_credit_card_clients` dataset from the course folder on Google drive. The data description is available at : https://www.kaggle.com/datasets/jishnukoliyadan/taiwan-default-credit-card-clients

1. Explore (5+5+10=20 points)
2. load the dataset. Use only the columns "LIMIT_BAL", "SEX", "EDUCATION", "MARRIAGE", and "AGE" among predictors. The target is "default payment next month".
3. identify the categorical features (with brief 1-3 sentence explanation), and
4. produce the pairwise scatter plot only for the numeric variables.
5. Prepare a pipeline to (30 points)
6. standardize the numeric attributes
7. expand the categorical attributes to columns of 0/1 variables
8. fit a `RandomForestClassifier` classifier
9. Search over the `max_depth` and `min_samples_leaf` parameters to find the best model per **balanced accuracy** metric. Use at least three different search strategies and discuss any differences you see in the results (≈ 150—200 words). (20 points)
10. Let's assume that the cost of missing a default (i.e., predicting non-default for a customer who ended up defaulting) is 10 times the cost of flagging a non-defaulter as defaulter. Let's further assume that the cost of correct predictions are 0. Use any one of the search strategies considered in the previous question to find the `RandomForestClassifier` that minimizes the cost. (20 points)
11. Collaboration statement: Who did you discuss while answering this homework (whether to get or to provide help)? What questions/topics did you discuss? Did you use any generative AI tool, such as ChatGPT? If so, provide your prompts. (10 points)

Note: No penalty for either side. While getting help in figuring out how to solve is OK, all answers should be produced by you.

If you did not collaborate with anyone simply declare so.

Question 1 - EDA

Importing Data

```
import pandas as pd
import seaborn as sns


from google.colab import drive
drive.mount('/content/drive')
data_folder =
'drive/Othercomputers/asus/MSBA/Fall/BA810Fall23Material/Slides/Data/'
```

```python
default =
pd.read_csv(data_folder+"default_of_credit_card_clients.csv", usecols
= [ "LIMIT_BAL", "SEX", "EDUCATION", "MARRIAGE", "AGE", "default
payment next month"])
display(default.head())
display(default.info())
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
   LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  default payment next
month
0      20000    2          2         1   24
1
1     120000    2          2         2   26
1
2      90000    2          2         2   34
0
3      50000    2          2         1   37
0
4      50000    1          2         1   57
0
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 6 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   LIMIT_BAL                   30000 non-null  int64
 1   SEX                         30000 non-null  int64
 2   EDUCATION                   30000 non-null  int64
 3   MARRIAGE                    30000 non-null  int64
 4   AGE                         30000 non-null  int64
 5   default payment next month  30000 non-null  int64
dtypes: int64(6)
memory usage: 1.4 MB
```

```
None
```

Categorical Features

In our dataset, we have 3 categorical variables:

1. SEX: Sex of customer | 1 = Male & 2 = Female

2. EDUCATION: Highest level of education of customer | 1 = graduate school; 2 = university; 3 = high school; 4 = others

3. MARRIAGE: Current marital status of customer | 1 = married; 2 = single; 3 = others

These three features are comprised of finite, discrete values, making them categorical. In order to represent these features in a way that is interpretable by models, we will need to encode these features.

Data Cleaning

```
# Checking for outlier values

display(default.describe())

# Replacing any EDUCATION values > 4 with 4
default['EDUCATION'] = default['EDUCATION'].map({1:1, 2:2, 3:3, 4:4,
5: 4, 6: 4})
```

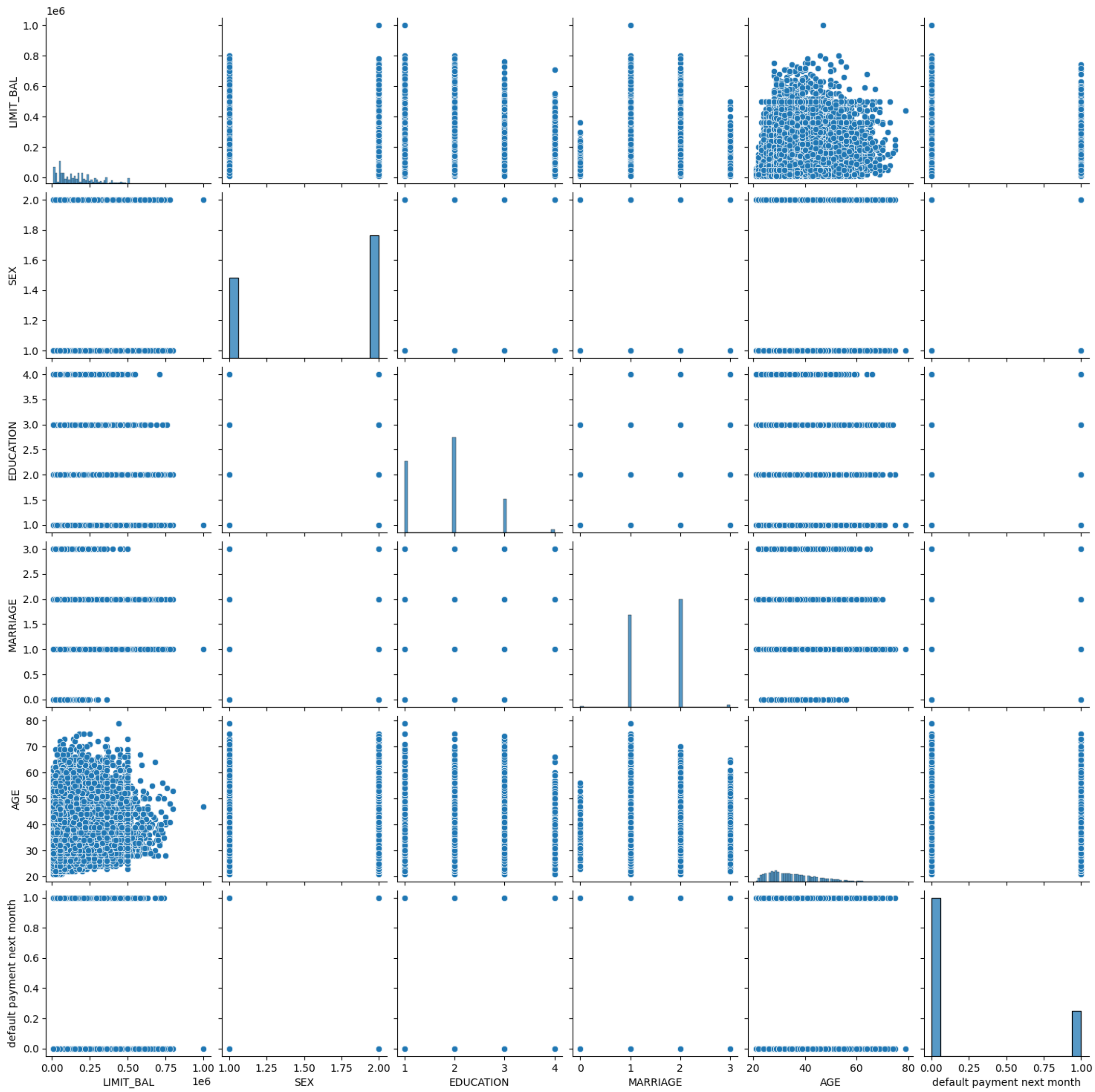
```
            LIMIT_BAL           SEX     EDUCATION      MARRIAGE
AGE  \
count    30000.000000  30000.000000  30000.000000  30000.000000
30000.000000
mean    167484.322667      1.603733      1.853133      1.551867
35.485500
std     129747.661567      0.489129      0.790349      0.521970
9.217904
min      10000.000000      1.000000      0.000000      0.000000
21.000000
25%      50000.000000      1.000000      1.000000      1.000000
28.000000
50%     140000.000000      2.000000      2.000000      2.000000
34.000000
75%     240000.000000      2.000000      2.000000      2.000000
41.000000
max    1000000.000000      2.000000      6.000000      3.000000
79.000000

       default payment next month
count                30000.000000
mean                     0.221200
std                      0.415062
min                      0.000000
```

```
25%                          0.000000
50%                          0.000000
75%                          0.000000
max                          1.000000
```

Replaced all EDUCATION values greater than the defined maximum (4 = others) with 4 to keep in line with the data dictionary.

```
sns.pairplot(default)
```

```
<seaborn.axisgrid.PairGrid at 0x7d4bc3be06d0>
```



Note: Including all variables (since there are only two numeric variables).

Observations:

- The classes are fairly imbalanced (defaults are only ~20%)

- There is a positive correlation between AGE and LIMIT_BAL, which is reasonable because older customer's are likely to have more establlished credit histories and borrowing capacity

Q2. Model Pipeline

```
# Train-Test Split

from sklearn.model_selection import train_test_split

X = default.drop("default payment next month", axis=1)
y = default["default payment next month"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((22500, 5), (7500, 5), (22500,), (7500,))
```

```
# Preprocessing Pipeline - Encoding and Scaling

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

from sklearn import set_config
set_config(display='diagram') # display pipeline diagram

cats = ["SEX", "EDUCATION", "MARRIAGE"]
nums = ["LIMIT_BAL", "AGE"]

preprocess_pipeline = ColumnTransformer([
        ("cat", OneHotEncoder(drop="first"), cats), # Using one-hot
encoding for categoricals
        ("num", StandardScaler(), nums), # Standardizing numerics
using Z-Transformation
    ])

X_train_transformed = preprocess_pipeline.fit_transform(X_train) #
transform columns
print(preprocess_pipeline.get_feature_names_out()) # check processed
column names
preprocess_pipeline # display pipeline
```

```
['cat__SEX_2' 'cat__EDUCATION_2.0' 'cat__EDUCATION_3.0'
 'cat__EDUCATION_4.0' 'cat__EDUCATION_nan' 'cat__MARRIAGE_1'
 'cat__MARRIAGE_2' 'cat__MARRIAGE_3' 'num__LIMIT_BAL' 'num__AGE']

ColumnTransformer(transformers=[('cat', OneHotEncoder(drop='first'),
                                 ['SEX', 'EDUCATION', 'MARRIAGE']),
                                ('num', StandardScaler(),
                                 ['LIMIT_BAL', 'AGE'])])
```

The preprocessing pipeline is working. We can proceed to modeling.

```
# Modeling

from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.metrics import balanced_accuracy_score

rf_pipe = make_pipeline(preprocess_pipeline, RandomForestClassifier())
# creating RF pipeline
rf_pipe.fit(X_train, y_train) # fitting RF model

y_train_pred = rf_pipe.predict(X_train)
print(f'Training Balanced Accuracy: {balanced_accuracy_score(y_train,
y_train_pred):.3f}') # measuring balanced accuracy of fitted model

rf_params = rf_pipe.get_params() # getting model params
rf_md = rf_params['randomforestclassifier__max_depth']
rf_msl = rf_params["randomforestclassifier__min_samples_leaf"]
print(f"Models parameters - max_depth = {rf_md} & min_sample_leaf =
{rf_msl}")

Training Balanced Accuracy: 0.697
Models parameters - max_depth = None & min_sample_leaf = 1
```

Q3 Hyperparameter Tuning

Grid Search

```
#Grid Search
from sklearn.model_selection import GridSearchCV
import numpy as np

param_grid = [ # values have been chosen based on external guide on
Random Forest tuning
    {'randomforestclassifier__max_depth': list(np.arange(1, 10000,
step=2000)) + [None], # ranges from 1 to 10000 in steps of 2000 + None
(no max depth)
     'randomforestclassifier__min_samples_leaf': [1, 2, 4]
     }
    ]
```

```python
# Print parameter grid
print('The parameter grid : ')
print(param_grid)

# Comparing all 18 combinations of these values using cross
validation:

grid_search = GridSearchCV(rf_pipe, param_grid, cv=3,
scoring='balanced_accuracy')
grid_search.fit(X_train, y_train)
print('\n\nThe best parameters are ', grid_search.best_params_)

grid_cv_res = pd.DataFrame(grid_search.cv_results_) # convert to
dataframe
grid_cv_res.sort_values(by='mean_test_score', ascending=False,
inplace=True)  # sort by CV balanced accuracy
# select only required columns
grid_cv_res.filter(regex = '(^param_|mean_test_score)', axis=1).head()
```

```
The parameter grid :
[{'randomforestclassifier__max_depth': [1, 2001, 4001, 6001, 8001,
None], 'randomforestclassifier__min_samples_leaf': [1, 2, 4]}]


The best parameters are  {'randomforestclassifier__max_depth': 4001,
'randomforestclassifier__min_samples_leaf': 1}

   param_randomforestclassifier__max_depth  \
6                                     4001
9                                     6001
3                                     2001
12                                    8001
15                                    None

   param_randomforestclassifier__min_samples_leaf  mean_test_score
6                                               1         0.527186
9                                               1         0.525730
3                                               1         0.525546
12                                              1         0.525392
15                                              1         0.524719
```

Results: Best Balanced Accuracy is lower than untuned model, but this is likely due to the model only training on 2/3rds of training set in each iteration.

Best Parameters:

- max_depth : 4001

- min_samples_leaf : 1

Random Search

```
#Random Search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint


param_grid = [ # same range as grid search
    {'randomforestclassifier__max_depth': list(np.arange(10, 10000,
step=1)) + [None], # all integers 10 to 10000 + None
     'randomforestclassifier__min_samples_leaf': randint(1,10)  # all
integers from 1 to 10 (inclusive)
     }
    ]

# Comparing 18 combinations values from defined ranges using cross
validation:

random_search = RandomizedSearchCV(rf_pipe, param_grid, n_iter=18, #
same as grid search combinations
                                   cv=3, scoring='balanced_accuracy')
random_search.fit(X_train, y_train)
print('\n\nThe best parameters are ', random_search.best_params_)

random_cv_res = pd.DataFrame(random_search.cv_results_) # convert to
dataframe
random_cv_res.sort_values(by='mean_test_score', ascending=False,
inplace=True)  # sort by CV balanced accuracy
# select only required columns
random_cv_res.filter(regex = '(^param_|mean_test_score)',
axis=1).head()
```

```
The best parameters are  {'randomforestclassifier__max_depth': 305,
'randomforestclassifier__min_samples_leaf': 1}

   param_randomforestclassifier__max_depth  \
11                                      305
10                                     7066
6                                      4632
8                                      1338
```

```
1                                   8889

   param_randomforestclassifier__min_samples_leaf  mean_test_score
11                                              1         0.526659
10                                              1         0.525031
6                                               1         0.524379
8                                               2         0.516036
1                                               2         0.515138
```

Results: mean_test_score is nearly identical to grid search.

Best Parameters:

- max_depth : 305

- min_samples_leaf : 1

Note: The above parameter combination is one was not one of the pairs tested in grid search. Random seach was able to tests parameter combinations that grid search wasn't able to due to the latters limited search space.

Bayesian Search

```
!pip install scikit-optimize  # install skopt

Collecting scikit-optimize
  Downloading scikit_optimize-0.9.0-py2.py3-none-any.whl (100 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/100.3 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╸━━━ 92.2/100.3 kB 2.9 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.3/100.3 kB 2.4
MB/s eta 0:00:00
ent already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.3.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-23.9.7-py3-none-any.whl (23 kB)
Requirement already satisfied: numpy>=1.13.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(1.23.5)
Requirement already satisfied: scipy>=0.19.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(1.11.3)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-
optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>scikit-optimize) (3.2.0)
```

```
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-23.9.7 scikit-optimize-0.9.0
```

```
#Baysian Search
from skopt import BayesSearchCV
from skopt.space import Integer

param_distribs = {
    'randomforestclassifier__max_depth': Integer(10, 10000), # Setting high upper limit to represent None max_depth
    'randomforestclassifier__min_samples_leaf': Integer(1, 10),
}

bayes_search = BayesSearchCV(
    rf_pipe, param_distribs, n_iter=18, # keeping number of iterations consistent with previous methods
    cv=3, scoring='balanced_accuracy', random_state=42)

bayes_search.fit(X_train, y_train)

# Checking results
bayes_res = pd.DataFrame(bayes_search.cv_results_)
bayes_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
bayes_res.filter(regex = '(^param_|mean_test_score)', axis=1).head()
```

```
   param_randomforestclassifier__max_depth  \
16                                    9998
12                                   10000
10                                      49
17                                      38
3                                     8126

   param_randomforestclassifier__min_samples_leaf  mean_test_score
16                                              1         0.525689
12                                              1         0.525248
10                                              1         0.524823
17                                              1         0.524648
3                                               3         0.512370
```

Results : mean_test_score is nearly identical to previous searches

Best Parameters: Different from default and grid search

- max_depth : 9998 (i.e. tending to max_depth = None)

- min_samples_leaf : 1

```
# Checking Test Balanced Accuracy

# Extracting tuned models
```

```
grid_rf = make_pipeline(preprocess_pipeline,
RandomForestClassifier(max_depth=4001, min_samples_leaf=1))
grid_rf.fit(X_train, y_train)
random_rf =  make_pipeline(preprocess_pipeline,
RandomForestClassifier(max_depth=305, min_samples_leaf=1))
random_rf.fit(X_train, y_train)
bayes_rf =  make_pipeline(preprocess_pipeline,
RandomForestClassifier(max_depth=9998, min_samples_leaf=1))
bayes_rf.fit(X_train, y_train)

# Making predictions
grid_preds = grid_rf.predict(X_test)
random_preds = random_rf.predict(X_test)
bayes_preds = bayes_rf.predict(X_test)
default_preds = rf_pipe.predict(X_test)
# Balanced Accuracies

print(f'Testing Balanced Accuracy - Default:
{balanced_accuracy_score(y_test, default_preds):.3f}')
print(f'Testing Balanced Accuracy - Grid:
{balanced_accuracy_score(y_test, grid_preds):.3f}')
print(f'Testing Balanced Accuracy - Random:
{balanced_accuracy_score(y_test, random_preds):.3f}')
print(f'Testing Balanced Accuracy - Bayes:
{balanced_accuracy_score(y_test, bayes_preds):.3f}')
```

```
Testing Balanced Accuracy - Default: 0.520
Testing Balanced Accuracy - Grid: 0.521
Testing Balanced Accuracy - Random: 0.519
Testing Balanced Accuracy - Bayes: 0.522
```

Conclusions

1. It appears that a very high max_depth (or None) + min_sample_leafs = 1 is the optimal hyperparameter set.

2. Only Bayesian search (and Grid to an extent) were capable of finding this pair in this case. Grid was able to find it since it was explicitly told to test this pair, but bayesian search was able to uncover this solution by itself when the upper limit was sufficiently high. Bayesian search was also able effectively to test values in a much larger range than Grid or Random search. Random search failed to detect optimal max_depth because it was not one selected in any of the 18 iterations.

3. Grid Search - useful if we need to test specific hyperparameter combinations, but must be programed explicitly (and thus can be sub-optimal)

4. Random Search - useful for searching a wider array of hyperparameter combinations, but can miss optimal solution if iterations are few

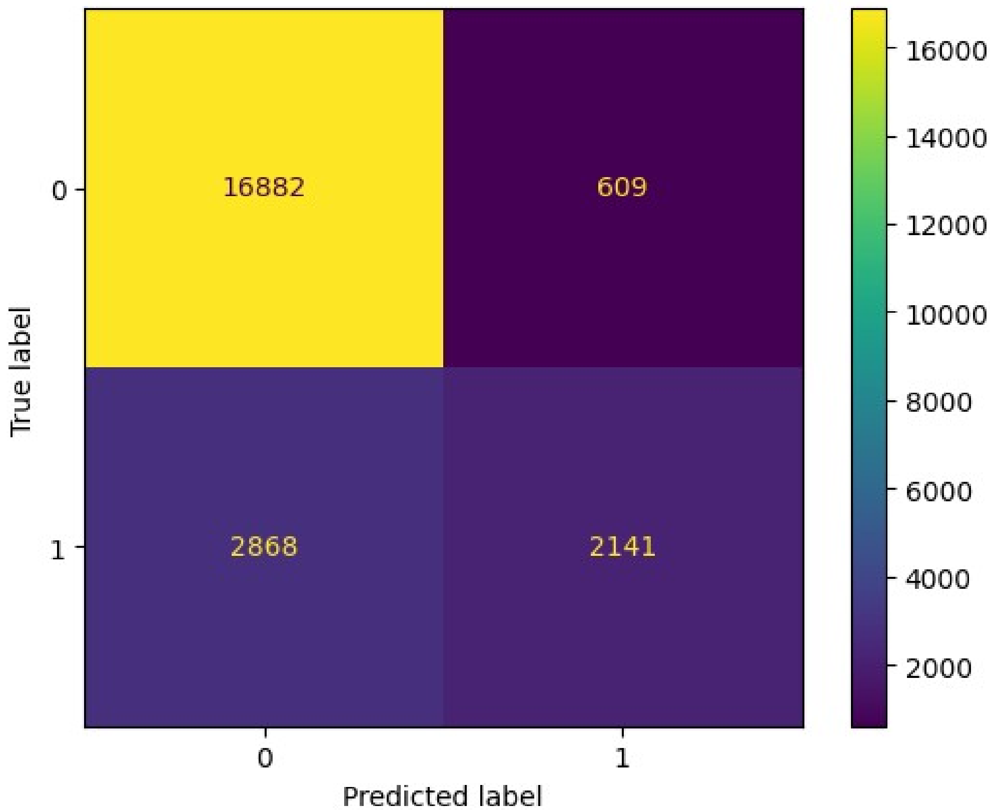
5. Bayesian Search - can find optimal parameter without explicit instructions, and can test a wide array of combinations. Though it cannot include None in search space (since it is not a numeric value), this can be remedied by using a a sufficiently high upper limit. If the discovered ideal max_depth is near the upper bound, it is an indicator that a max_depth of None should be considered and tested.

Bayesian search appears to be the most effective and efficient tuning method in this case.

Q4 Cost-sensitive Tuning

```
# Displaying Training Confusion matrix for reference

traincm = confusion_matrix(y_train, y_train_pred)
ConfusionMatrixDisplay(traincm)

disp = ConfusionMatrixDisplay(confusion_matrix=traincm,
                              display_labels=rf_pipe.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7d4bc3b1fb80>
```




```
from sklearn.metrics import make_scorer, confusion_matrix,
ConfusionMatrixDisplay

# creating scoring function

def cost_calc(y_true, y_pred):
  cm = confusion_matrix(y_true, y_pred)
  return cm[1,0] * 10 + cm[0,1] * 1
  # return total false negative cost + total false positive cost

# making scorer object
cost_scorer = make_scorer(cost_calc)

# calculating test cost of default model
print(f"Default Total Cost = {cost_calc(y_test,
rf_pipe.predict(X_test))}")

Default Total Cost = 14476

# Bayesian search

param_distribs = {
    'randomforestclassifier__max_depth': Integer(10, 10000), # Setting
high upper limit to represent None max_depth
    'randomforestclassifier__min_samples_leaf': Integer(1, 4)}
```

```python
bayes_search_cost = BayesSearchCV(
    rf_pipe, param_distribs, n_iter=18, # keeping number of iterations
consistent with previous methods
    cv=3, scoring=cost_scorer, random_state=42)

bayes_search_cost.fit(X_train, y_train)

# Checking results
bayes_res_cost = pd.DataFrame(bayes_search_cost.cv_results_)
bayes_res_cost.sort_values(by="mean_test_score", ascending=True,
inplace=True)
bayes_res_cost.filter(regex = '(^param_|mean_test_score)',
axis=1).head()
```

```
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/
optimizer.py:449: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimizer.py:4
49: UserWarning: The objective has been evaluated at this point
before.
  warnings.warn("The objective has been evaluated "
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimizer.py:4
49: UserWarning: The objective has been evaluated at this point
before.
  warnings.warn("The objective has been evaluated "
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimizer.py:4
49: UserWarning: The objective has been evaluated at this point
before.
  warnings.warn("The objective has been evaluated "
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimizer.py:4
49: UserWarning: The objective has been evaluated at this point
before.
  warnings.warn("The objective has been evaluated "
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimizer.py:4
49: UserWarning: The objective has been evaluated at this point
before.
  warnings.warn("The objective has been evaluated "
```

```
   param_randomforestclassifier__max_depth  \
10                                    1023
4                                     7998
3                                     8126
0                                     4107
6                                     6175

   param_randomforestclassifier__min_samples_leaf  mean_test_score
10                                              1     14613.333333
4                                               2     15583.000000
```

```
3                                                2     15618.333333
0                                                3     15913.333333
6                                                3     15968.000000
```

```
# Grid Search

param_grid = [ # values have been chosen based on external guide on
Random Forest tuning
    {'randomforestclassifier__max_depth': list(np.arange(1, 10000,
step=2000)) + [None], # ranges from 10 to 10000 in steps of 2000 +
None (no max depth)
     'randomforestclassifier__min_samples_leaf': [1, 2, 4]
     }
    ]


grid_search_c = GridSearchCV(rf_pipe, param_grid, cv=3,
scoring=cost_scorer)
grid_search_c.fit(X_train, y_train)
print('\n\nThe best parameters are ', grid_search_c.best_params_)

grid_cv_res_c = pd.DataFrame(grid_search_c.cv_results_) # convert to
dataframe
grid_cv_res_c.sort_values(by='mean_test_score', ascending=True,
inplace=True)  # sort by CV balanced accuracy
# select only required columns
grid_cv_res_c.filter(regex = '(^param_|mean_test_score)',
axis=1).head()
```

```
The best parameters are  {'randomforestclassifier__max_depth': 1,
'randomforestclassifier__min_samples_leaf': 1}

   param_randomforestclassifier__max_depth  \
9                                     6001
15                                    None
3                                     2001
12                                    8001
6                                     4001

   param_randomforestclassifier__min_samples_leaf  mean_test_score
9                                                1     14554.000000
15                                               1     14558.000000
3                                                1     14567.000000
12                                               1     14590.333333
6                                                1     14598.333333
```

```
# Checking Optimal Models' Test Cost

best_cost_grid =  make_pipeline(preprocess_pipeline,
```

```
RandomForestClassifier(max_depth=6001, min_samples_leaf=1))
best_cost_grid.fit(X_train, y_train)

print(f"Optimal Total Cost - Grid = {cost_calc(y_test,
best_cost_grid.predict(X_test))}")
```

```
Optimal Total Cost - Grid = 14449
```

Optmial Hyperparameters for Minimizing Cost: max_depth = 6001 | min_sample_leaf = 1

Optimal Cost = 14449

Q5 - Collaboration

I did not collaborate with anyone